Package: projectData.Seurat (via r-universe)

November 28, 2026

Title 'projectData' Adapter for Disk-Backed 'Seurat' Objects

Version 0.1.0

Description A loader for the 'projectData' adapter framework. It restores a saved 'Seurat' v5 object whose assay layers are 'BPCells' on-disk matrices from an installed data package, on first access, without attaching 'Seurat' or 'projectData' to the search path and without writing to the installed package. For data package authors, it saves such an object with its lazy 'BPCells' operations and shared backing preserved, and registers it in the package's 'projectData' manifest. It relies on the 'SeuratObject' v5 assay layer architecture, which the tested 'Seurat' v5 and v6 releases share.

License MIT + file LICENSE

URL https://github.com/blaserlab/projectData.Seurat

BugReports https://github.com/blaserlab/projectData.Seurat/issues

Depends R (>= 4.1.0)

Imports methods, projectData (>= 0.1.0), stats, utils

Suggests BPCells, callr, DelayedArray, fs, knitr, Matrix, pkgload, rmarkdown, Seurat, SeuratObject (>= 5.0.0), testthat (>= 3.2.0), withr (>= 2.5.0)

VignetteBuilder knitr

Remotes github::bnprks/BPCells/r@v0.3.1

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Encoding UTF-8

Roxygen list(markdown = TRUE)

Repository https://blaserlab.r-universe.dev

Date/Publication 2026-11-28 19:42:18 UTC

RemoteUrl https://github.com/blaserlab/projectData.Seurat

RemoteRef HEAD

RemoteSha 73d67bdff6ab6af1198a5430d13a4a60aaf402f7

Contents

projectData_loaders 2
save_seurat_object 3
seurat_storage_report 7
write_seurat_manifest 8

Index **10**

`projectData_loaders` *Loader declaration for disk-backed Seurat objects*

Description

`projectData_loaders()` is the adapter export that projectData looks for when a manifest names `AdapterPackage: projectData.Seurat`. It returns one `projectData::data_loader()` with id `projectData.Seurat::seurat`. projectData loads this package's namespace (it does not attach it) the first time a resource that needs the loader is accessed.

Usage

```
projectData_loaders()
```

Details

Each resource is a directory, `extdata/<name>`, written by `save_seurat_object()`:

```
object.rds     the Seurat object, saved with saveRDS()
backing.dcf    index of the backing directories
bpcells/<id>/  one BPCells matrix directory per physical matrix
```

The leaves of every layer's BPCells operation graph in `object.rds` carry paths relative to the resource. The loader retargets those leaves to the resource's absolute location with `BPCells::all_matrix_inputs<-`, which restores lazy operation graphs and shared backing exactly. Nothing from the file is evaluated, nothing is written or copied, and the working directory is never involved, so the installed files can be read-only.

- `validate` checks the directory structure and path containment only, including that every directory listed in `backing.dcf` exists and is a BPCells matrix, so a damaged installation is caught at registration. It does not read `object.rds` and never opens a matrix.
- `open` errors, never returning a partial object, if any backing directory is missing, unlisted, unreferenced, escaping the resource, or of unsupported storage.

Value

A `projectData_loader` object.

What the restored object references

The returned object reads its BPCells layers lazily from the installed resource directory. Those files must stay installed for as long as the object is in use; the loader never deletes or copies them, and `projectData::data_detach()` does not invalidate a retained object.

See Also

`projectData::data_loader()`, `projectData::data_loader_adapter()`

---

save_seurat_object *Save a disk-backed Seurat object into a data package source tree*

Description

`save_seurat_object()` is a **package-authoring** operation. One call writes everything a data package needs to ship a Seurat object whose layers are backed by BPCells directories, so that `projectData::data_attach()` opens it through the `projectData.Seurat::seurat` loader:

Usage

```
save_seurat_object(object, name, package_dir = ".", overwrite = FALSE)
```

Arguments

`object`	A Seurat object with at least one BPCells directory-backed layer.
`name`	A single syntactically valid R name, not starting with a dot. It names the resource directory, the placeholder, the manifest resource and the object that `projectData::data_attach()` provides.
`package_dir`	Root of an existing R package *source* directory.
`overwrite`	Replace an existing saved dataset and its placeholder?

Details

```
my.datapkg/
|-- data/<name>.rda                 placeholder (written here)
`-- inst/
    |-- extdata/<name>/
    |   |-- object.rds              the Seurat object (written here)
    |   |-- backing.dcf             index of the backing directories
    |   `-- bpcells/<id>/           one copy of each physical matrix
    `-- projectData/manifest.dcf    entry for <name> (created or updated)
```

Nothing is written to the working directory (unless `package_dir` is it), `R/data.R` is never edited, and neither `object` nor its backing directories are modified.

Value

Invisibly, a list with `name`, `package`, `directory`, `placeholder`, `manifest`, `manifest_status`, `replaced`, `removed_stale`, `layers` (one row per layer: `assay`, `layer`, `kind`, `n_stores`, `stores`, `values_compared`) and `backing` (one row per physical store: `id`, `source`, `path`, `files`, `bytes`, `layers`).

Side effects

Under `package_dir` only: creates `inst/extdata/<name>/` (backing copies and `object.rds`), `data/<name>.rda` and the manifest entry, creates the `inst/` and `data/` directories if they do not exist, and creates and removes a hidden `.projectData.Seurat-stage-*` directory. With `overwrite = TRUE` it replaces the previous resource and placeholder, and may remove stale manifest entries of this loader. It prints a short message summarizing what was written and whether `R/data.R` lacks a topic for `name`. Calling it again with the same arguments and `overwrite = TRUE` leaves the package in the same state (without `overwrite` it is refused, with class `projectData.Seurat_exists`).

Storage is preserved, never converted

In current Seurat, layers are not separate matrices. `data` is usually a *lazy* BPCells chain (scale, log1p, row selection, ...) over the same directory as `counts`, and `scale.data` can be another chain on top. So the unit of storage is the **physical backing directory**, found through `BPCells::all_matrix_inputs()` (the leaves of each operation graph), not the layer, and a layer's top-level class does not show whether it is on disk.

- Every distinct directory is copied **once**, however many layers use it.
- The operation graph of each layer is kept: only the directory of each leaf is retargeted, to a path relative to the resource, with BPCells' `all_matrix_inputs<-`. Layers that shared a directory still share it.
- In-memory layers (R matrices, or BPCells graphs over in-memory leaves) are stored inside `object.rds` as they are.
- No lazy operation is ever evaluated, no matrix is densified or converted, and nothing is normalized or scaled.

`SeuratObject::SaveSeuratRds()` is deliberately **not** used: it records one (path, reopen-as-plain-matrix) pair per layer, so a lazy `data` layer comes back as the raw counts, with no warning (verified; see `AGENTS.md`). The object is written with `saveRDS()`, which is the last step of `SaveSeuratRds()` too.

What is accepted, and what is refused before writing

At least one layer must be backed by a BPCells `MatrixDir`. Refused, with an error of class `projectData.Seurat_unsupported_storage` listing every offending assay and layer: leaves other than `MatrixDir` (HDF5, anndata, BPCells in-memory matrices, ...), `DelayedArray` and other matrix classes, BPCells matrices inside a v3 `Assay`, spatial images, backing directories that do not exist or are not BPCells directories, nested backing directories, and BPCells or `DelayedArray` matrices hidden outside the assay layers (in the `misc`, `tools` or `commands` of the object, or the `misc` of an assay or reduction), which would be stored with the author's paths and dangle after installation. An

object that is not disk-backed at all is refused too; ordinary in-memory objects belong in an ordinary `.rda` dataset. Backing directories inside the resource being replaced are also refused, since replacing it would delete them under the object.

Not examined: file paths held as plain text anywhere in the object (they cannot be told from other strings), and anything in a slot other than those listed. Such strings are stored as they are and are not rewritten.

Filesystem containment

The backing directories named by the object are the *external sources being imported*; they are read, never written, moved or deleted. A source directory may itself be reached through a symbolic link (it is judged by its resolved location), but **inside it** only regular files are accepted: any symbolic link (to the inside, the outside, or nowhere) and any nested directory is refused. This is deliberate: copying follows links, so a link would put a file from elsewhere into the resource. Entries are examined when the object is inventoried, again immediately before each copy, and the copies are examined afterwards. A source that contains the package being written to is refused. Destinations are plain `bpcells/<id>` names generated from assay and layer names and must resolve inside the staged resource; all writing happens in a hidden stage directory in the package root. FIFOs and device files cannot be recognized from base R and are not detected.

Verification before committing

The staged resource is checked, and nothing is committed on failure. Three kinds of check are made, and they are not equally strong:

- **Structural**: the loader's `validate` passes; the resource contains only `object.rds`, `backing.dcf` and one directory per physical store; file names and **sizes** equal the originals; the originals' file names, sizes and times are unchanged; the resource opens through the loader without writing; cells, metadata, assays, reductions, layer names, classes, dimensions and dimnames are equal.
- **Numerical**: values are compared through the restored object. A layer of at most 1e6 entries is compared completely; a larger one by four columns spread across it, so a large matrix is never read in full. The result (`"complete"` or `"sampled (4 of N columns)"`) is returned per layer in `layers$values_compared`. A sample is not a proof about unsampled columns.
- **Operation graph**: the BPCells operation graph of every layer is compared as plain nested values (every node's class and slots, i.e. operations and their parameters, dimnames, flags) with the leaf directories left out, plus the lines of the print method; which physical store every leaf uses is compared separately, so sharing is covered. This uses generic S4 introspection, not BPCells internals, but it is a comparison of two representations of the same package version, not a formal equivalence of computations.

File **checksums are not computed** or stored. Hashing every backing matrix on each save and each load would cost a full read of data that is meant to stay on disk, and the copies are made by the operating system's file copy, with sizes verified. Authors who need byte-level assurance can compare trees themselves (for example with `tools::md5sum()`); `R CMD build` and installation do not alter file contents.

Overwriting, failure and the manifest

Existing targets are replaced only with `overwrite = TRUE`, and only if this package can show that it owns them. A directory that merely contains an `object.rds` or a `backing.dcf` is *not* owned. A

resource directory is owned when it is empty, or its content is exactly `object.rds`, `backing.dcf` and `bpcells/<id>/` directories (no symbolic links, nothing else), any manifest entry for the name belongs to this loader and agrees with it (Source `extdata/<name>`, Objects `<name>`), and either

- it passes the loader's validation (a complete resource), or
- it is damaged but at least two independent pieces of evidence remain: a structurally valid `backing.dcf`, a manifest entry of this loader, a recognized placeholder.

Anything else (one piece of evidence, a manifest entry owned by another loader or disagreeing with the directory, unexpected content, a symbolic link or file where the directory should be) is an error of class `projectData.Seurat_not_owned` that says why, and nothing is written. A `data/<name>.rda` is replaced only if it holds a placeholder written by this package; one holding anything else is never replaced. There is no `force` option. A name used by another loader, or declared as an object by another resource, is an error. The manifest is reconciled as by `write_seurat_manifest()`.

New files are created in a hidden `.projectData.Seurat-stage-*` directory in the package root, then renamed into place with the previous files kept in the stage's `backup/` directory; if any step fails, or R is interrupted, the new files are removed and the old ones moved back, and the stage is deleted. If restoring fails too, the stage is **kept** and the error gives the exact location of the backups. Backing files of the author's objects are never deleted; after a successful overwrite the *previous resource* is deleted, so an object still reading from it (only possible from the source tree) stops working. This is not a filesystem transaction: renames are atomic one at a time, but the set of them is not, and the manifest is written last by projectData. If the R process is killed (or the machine loses power) during the commit, the package can be left with the stage directory, and possibly with some old files in `backup/` and some new ones in place. The stage is safe to delete once the package is in the state you want; it holds the previous files when a commit was cut short.

Documentation

Document the dataset with a roxygen2 block in `R/data.R` ending in the quoted name. This function only tells you when no such topic is found.

Conditions

Every error has a class, so it can be handled with `tryCatch()`. Besides `"error"`, the classes are:

Class	Raised when
`projectData.Seurat_package_dir_error`	`package_dir` is not an existing package source directory, or is not writable
`projectData.Seurat_dependency_error`	SeuratObject or BPCells is missing or incompatible
`projectData.Seurat_unsupported_storage`	the object holds storage that is refused (see above)
`projectData.Seurat_backing_in_target`	the object's backing lies inside the resource being replaced
`projectData.Seurat_exists`	the name exists and `overwrite = FALSE`
`projectData.Seurat_not_owned`	the existing resource or placeholder cannot be shown to be this package's
`projectData.Seurat_conflict`	the name is used by another loader's manifest entry
`projectData.Seurat_manifest_error`	the existing manifest is unreadable or invalid
`projectData.Seurat_verification_error`	the staged copy failed verification, or an original changed while saving

Errors other than `projectData.Seurat_package_dir_error` also inherit `projectData.Seurat_save_error`.
All of them are raised before anything is written, or after the previous state has been restored.

See Also

write_seurat_manifest(), seurat_storage_report(), projectData_loaders(); vignette("authoring-seurat-da
package = "projectData.Seurat")

Examples

```
## Not run:
save_seurat_object(seu, "seu_main", package_dir = ".")

## End(Not run)
```

`seurat_storage_report` *Report how the layers of a Seurat object map onto physical BPCells storage*

Description

`seurat_storage_report()` answers one question about an object, either one you are about to save or one restored from a data package: **how many logical assay layers are there, and how many physical BPCells matrices do they actually read from?** In current Seurat, `data` and `scale.data` are usually lazy chains of operations over the same BPCells directory as `counts`, so several layers can share one store (see the "Storage" vignette).

Usage

```
seurat_storage_report(object)
```

Arguments

`object`	A Seurat object.

Details

It is read-only and cheap. It inspects the BPCells operation graph of each layer (through `BPCells::all_matrix_inputs()`), lists the entries of each backing directory to measure file sizes, and never reads matrix values, evaluates a lazy operation, or writes anything. It does not check whether the object can be saved; `save_seurat_object()` does that and refuses unsupported storage.

Value

A list of class `projectData.Seurat_storage_report` with

- `layers`: a data frame with one row per assay layer: `assay`, `layer`, `representation`, `operations`, `backing` (the store labels the layer reads, comma-separated; empty for in-memory layers);
- `stores`: a data frame with one row per distinct physical directory: `store` (a label, the directory name made unique), `path`, `files`, `bytes`, `layers` (the `assay/layer` entries that read it);
- `n_layers` and `n_stores`.

Printing shows the two tables and a one-line summary. The column names and the four representation values are part of the documented interface.

Representations

The `representation` column has one of four values.

Value	Meaning
`"BPCells"`	Reads a BPCells directory with no operation queued other than naming rows and columns
`"Lazy BPCells"`	Reads a BPCells directory through queued operations (scaling, `log1p`, row selection, ...) that are computed
`"In memory"`	An ordinary R matrix, or BPCells operations over in-memory data only
`"Unsupported"`	Any other class of matrix (for example `DelayedArray`) or BPCells storage other than a directory (for exam

`operations` counts the BPCells operation nodes above the stores, excluding the dimnames-only wrapper that Seurat puts on every layer.

Sizes

`files` and `bytes` in `stores` are **measured sizes of the files in each backing directory** at the time of the call. They are unrelated to the logical dimensions of the layers (a lazy `data` layer has the dimensions of a full expression matrix but adds no bytes of its own). `bytes` is `NA` if the directory does not exist (for example after the data package was removed).

See Also

`save_seurat_object()`

Examples

```
## Not run:
seurat_storage_report(seu)

## End(Not run)
```

`write_seurat_manifest` *Register saved Seurat resources in a data package's manifest*

Description

`write_seurat_manifest()` is a **package-authoring** operation. It finds resources written by `save_seurat_object()` (or by hand, in the same layout) under `inst/extdata/` of a package source tree and creates or updates `inst/projectData/manifest.dcf` so that projectData opens each one through the `projectData.Seurat::seurat` loader. The manifest is written at authoring time only; installed packages never generate one.

Usage

```
write_seurat_manifest(package_dir = ".")
```

Arguments

package_dir	Root of a data-package *source* directory (the one with DESCRIPTION).

Details

A directory directly below inst/extdata/ is a candidate when it contains object.rds. It is registered only if it passes the loader's own structural check (projectData_loaders()) **and** has a matching data/<name>.rda placeholder, recognized structurally (the file is parsed, never loaded, so nothing in it can run). A .rda that holds a real dataset is never treated as a placeholder. No Seurat object is read and no matrix is opened.

Value

Invisibly, a data frame with one row per resource considered: resource, status ("added", "updated", "unchanged", "removed", "skipped" or "conflict") and detail, with attributes manifest and written. Skipped resources and conflicts also raise warnings of class projectData.Seurat_skipped and projectData.Seurat_conflict.

Reconciliation

Entries whose Loader is projectData.Seurat::seurat are owned by this package and are reconciled with what is on disk:

Existing entry	Action
Owned, valid directory and placeholder	kept, or updated where it differs
Owned, directory gone	removed as stale
Owned, placeholder gone	removed as stale
New valid directory with placeholder	added
Any other loader or adapter	kept unchanged, with all its fields
Resource or object name used by another adapter	**conflict**: reported, never taken over
Owned, directory exists but is invalid	kept; the call fails, as the manifest would be invalid

Only manifest entries ever change: **saved directories and placeholders are never deleted**. A hand-written Title or Options is kept. The complete proposed manifest is validated and written atomically by projectData::data_manifest_write(); a failure leaves the previous manifest in place. Nothing is rewritten when nothing changed, so a second call is a no-op. If no entry remains, the manifest file is removed.

See Also

save_seurat_object(), projectData::data_manifest_write()

Examples

```
## Not run:
write_seurat_manifest(".")

## End(Not run)
```

Index

projectData::data_attach(), *3*
projectData::data_loader(), *2*, *3*
projectData::data_loader_adapter(), *3*
projectData::data_manifest_write(), *9*
projectData_loaders, 2
projectData_loaders(), *7*, *9*

save_seurat_object, 3
save_seurat_object(), *2*, *7–9*
seurat_storage_report, 7
seurat_storage_report(), *7*

write_seurat_manifest, 8
write_seurat_manifest(), *6*, *7*